VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Filip Šabacký

Bakalářská práce

Vedoucí práce: Ing. Marek Běhálek, Ph.D.

Ostrava, 2021

Abstrakt

Tato bakalářská práce popisuje průběh individuální odborné praxe vykonané ve společnosti KVADOS, a.s. Cílem práce bylo vytvoření vlastního frameworku pro automatické testování aplikace myTEAM®. Framework vychází z technologie zvané Selenium a má za úkol umožnit jednoduchou a optimalizovanou simulaci obsluhy aplikace myTEAM®. Součástí je také srovnání automatického a manuálního testování s následným zaměřením přímo na testování uživatelského rozhraní. Dále je v práci obsažen důkladný rozbor a popis vytvoření frameworku a jsou zde také rozebrány všechny hlavní problémy, které se objevily při vývoji. Na základě zjištěných informací je obsažen také následný plán dalšího rozvoje tohoto testovacího nástroje.

Klíčová slova

automatické testování; automatické testování UI; Selenium; myTEAM®; vytvoření vlastního frameworku pro automatické testování

Abstract

This bachelor thesis describes the course of individual professional practice performed in the company KVADOS, a.s. The aim of the work was to create your own framework for automatic testing of the myTEAM® application. The framework is based on a technology called Selenium and aims to enable a simple and optimized simulation of the operation of the myTEAM® application. It also includes a comparison of automatic and manual testing with a subsequent focus directly on user interface testing. Furthermore, the work contains a thorough analysis and description of the creation of the framework, and it also discusses all the main problems that emerged during development. Based on the information obtained, a subsequent plan for the further development of this testing tool is also included.

Keywords

automatic testing; automatic UI testing; Selenium; myTEAM®; creating your own framework for automatic testing

Poděkování

Rád bych poděkoval firmě KVADOS, a.s., která mi umožnila pracovat na této zajímavé bakalářské práci a celému týmu, který se podílí na vývoji aplikace myTEAM®. Zvláštní poděkování patří mému konzultantovi Ing. Antonínu Vaněčkovi, který mi vždy vyšel vstříc a se vším pomohl. V neposlední řadě chci poděkovat také mému vedoucímu práce Ing. Marku Běhálkovi, Ph.D.

Obsah

Seznam použitých zkratek a symbolů

CSS	–	Cascading Style Sheets
HTML	–	Hypertext Markup Language
DMS	–	Document Management System
DOM	–	Document Object Model
UI	–	User Interface
GUID	–	Globally Unique Identifier

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato bakalářská práce popisuje vytvoření vlastního frameworku pro automatické testování aplikace myTEAM®. Framework má umožnit jednoduchou a optimalizovanou simulaci obsluhy této aplikace a bude realizován na již existujícím řešením zvaném Selenium. V práci je představena firma KVADOS, a.s., u které tato práce vznikla.

Pro pochopení dané problematiky je v této práci obsažen teoretický rozbor testování webových aplikací. Jsou zde vysvětleny základní principy jak manuálního, tak hlavně automatického testování. Primárně je rozebrána problematika UI testování. Tento rozbor také ukazuje, proč nám nestačí pouze manuální testování a proč tedy vznikl nástroj k automatické obsluze, která aplikaci otestuje.

V další části jsou popsané hlavní věci, které bylo nutné zohlednit při tvorbě nového frameworku. Jedná se hlavně o speciality, díky kterých je aplikace myTEAM® jedinečná. Na to plynule navazuje vylíčení hlavních problémů, které se objevily při vývoji a bylo je nutné vyřešit.

V neposlední řadě nechybí zamyšlení nad dalším rozvojem tohoto nástroje pro automatické testování nebo nad jeho optimalizací a nechybí ani celkové zhodnocení celé práce.

1.1 Představení řešené problematiky

V dnešní době existuje celá řada aplikací, které se zabývají různou problematikou. Pokud zákazník něco potřebuje, často nalezne velké portfólio firem, které mu dokáží splnit jeho požadavky. Rozhodujícím faktorem dnes nebývá jen cena dané aplikace, popřípadně nějakého programu nebo nástroje, ale velký důraz je kladen také na jednoduchost, přehlednost a bezchybovost. Z tohoto důvodu stále více firem začíná investovat do testování svých aplikací. Nikdo dnes nechce v tomto zrychleném světě čekat, než se na obrazovce po dlouhé prodlevě konečně zobrazí požadovaný výsledek, vykoná se nějaká operace nebo ještě hůře, kdy program skončí chybou. Takové věci by se k žádnému uživateli nikdy neměly dostat, jelikož to velice snižuje kredit dané aplikace.

V rámci vývoje je prakticky nemožné vyvinout složitou aplikaci bez jediné chyby. Klíčové je, v jaké fázi vývoje se chyby podaří odhalit a kolik času zabere jejich odstranění. Častokrát se stane,

že programátor změní nějakou malou část aplikace, která vypadá po této změně v pořádku. Jeho zásahem ale mohl pokazit nebo poškodit úplně jinou část aplikace. U rozsáhlejších aplikací není v silách programátora, aby po jakékoliv změně otestoval celý program a nezapomněl na jedinou kombinaci, která by mohla nastat. Po delším čase, který stráví nad řešením daného problému, bývá obvyklé, že již nedokáže sám objektivně provést testování. V podstatě bývá zaslepen svou dosavadní prací. Proto v praxi je na testování vyčleněn tým lidí, který bezchybovost zodpovědný.

Tento tým se může zabývat automatickým nebo manuálním testováním. Manuální nebo jinak nazývané ruční testování znamená, že daný tester bude aplikaci testovat v reálném čase. Bude se snažit co nejlépe napodobit koncového uživatele, který s daným programem nebo aplikací bude pracovat. U takového testování je důležité být velice soustředěný a mít znalost dané problematiky. Jakákoliv nepozornost by mohla způsobit, že se k zákazníkovi dostane chybný software. Větší programy mívají zpravidla velké množství složitých kombinací, které někdy ani není možné jednoduše nasimulovat, případně je tato simulace zdlouhavá. Proto dnes bývají čím dál častěji dva oddělené týmy, kdy má jeden z nich na starost automatické testování a druhý tým bude naopak testovat pouze manuálně.

Obrázek 1.1: UI testování[1]

V případě automatického testování se tento tým snaží vyvinout imaginárního uživatele, který bude provádět stejné akce jako reálný uživatel. Nespornou výhodou pak bývá rychlost a znovupoužitelnost takových testů. Problémem naopak může být, že tato simulovaná obsluha vidí pouze to, pro co byla naprogramována. Nemá žádnou umělou inteligenci, která by předpovídala, co by se dále mohlo stát. Proto je důležité se snažit vytvořit co nejvíce hloupého uživatele neboli hloupý test, který se bude snažit všemi způsoby aplikaci pokazit.

Pokud se daný blok aplikace stále vyvíjí a zároveň na něho již existují automatické testy, bývají

častokrát problémy s údržbou těchto testů. I malá změna může mít za následek, že bude nutné přepsat také automatický test. Nejlepší variantou by bylo, kdyby se daná část aplikace vytvořila a dále se již nerozvíjela.

Jak už bylo zmíněno, hlavní výhodou manuálního testování je, že reálný člověk může vidět širší kontext dané problematiky. Může se tak zaměřit na určité části, které otestuje důkladněji. Některé chyby pak může zkušený tester odhalit dokonce rychleji. Nevýhodou naopak bývá, že toto testování je zdlouhavé a neefektivní, díky toho také finančně nákladné. Je potřeba myslet na to, že při každém testování je potřeba ověřit opět celou sadu kombinací. K tomu by testerům měla pomoct analýza, ve které by tyto kombinace měly být popsány.

V ideálním případě je nejlepší testovat jak automaticky, tak i ručně. Obě metody mají své výhody, ale také nevýhody, ty jsou podrobněji rozebrány a následně také porovnány na reálných případech v kapitole zaměřené na manuální testování 4 nebo automatické 5.

Kapitola 2

Charakteristika firmy a pracovního zařazení studenta

V této kapitole je představena firma KVADOS, a.s., její portfólio produktů a detailněji je rozebrán produkt myTEAM®, kvůli kterého tato práce vznikla.

2.1 Historie firmy

V roce 1992 byla Miroslavem Hampelem založena firma HANAS (zkratka slov HAmpel NAbízí Služby). Hned vzápětí, konkrétně o rok později, byla na jejich základech v roce 1993 založena také firma KVADOS (neboli KVAlitní DOdavatelské Služby). Měla cílit na velkou škálu zákazníků a poskytovat nejen kvalitní software a hardware, ale také vyvíjet softwarové řešení zákazníkům přímo na míru. [2]

Obrázek 2.1: Logo firmy[3]

2.2 Současnost

Firma KVADOS, a.s. pochází z Ostravy, kde sídlí dodnes. Zaměstnává přes 150 zaměstnanců a má velké portfólio vlastních produktů. Mezi zaměstnanci se nachází také studenti vysokých škol, kteří

se aktivně zapojují do rozvoje společnosti. Mimo jiné firma disponuje vlastním datovým centrem, díky němuž může zákazníkům nabídnout kompletní řešení na klíč.

Ze svých produktů může firma nabídnout například pokladní systém myCASH®, který obsahuje také podporu pro plánování spotřebitelských akcí nebo věrnostní program pro zákazníky. [4]

Pro logistické potřeby pak slouží řešení mySTOCK®, které umí řídit procesy a pracovníky ve skladech. Mezi výhody tohoto produktu také patří podpora mobilních zařízení. [5]

Produkt myAVIS® pomůže hlavně obchodníkům. Nabízí mobilní aplikaci, která například zajistí celý proces ohledně obchodní schůzky od její přípravy až po celkové vyhodnocení. Umí také řídit marketingové a distribuční procesy. [6]

Důležitým produktem je pak myVENTUS®, který vznikl již na počátku založení firmy. I přes svůj vysoký věk je ale u zákazníků stále oblíbený. Neustále se rozvíjí a vývojáři dynamicky reagují také na změny v legislativě. Nabízí mnoho modulů, například pro obchod a marketing, logistiku nebo ekonomiku. Výhodou také je, že podporuje multilicence a je přeložen do několika jazyků. [7]

2.3 myTEAM®

Pro kompletní přechod firem do digitálního prostředí slouží řešení nazvané myTEAM®. Celá aplikace je rozdělena do jednotlivých modulů, které na sobě nejsou závislé a zákazníci si je můžou kupovat samostatně dle vlastního výběru. Mezi hlavní moduly patří úkoly, DMS, smlouvy nebo objednávky. Obsahuje také nespočet procesů, které řídí všechny potřebné úkony digitálně a pomáhá s automatizací všech procesů. Klientská část aplikace je single page, která se spouští v internetovém prohlížeči. Jsou zde využity technologie HTML, CSS, JavaScript a Angular. Serverová část je pak napsaná v technoloigii .NET a celá aplikace běží jako služba v prostředí Windows. [8]

Podrobnější seznámení s produktem, jeho obsahem a jeho architekturou je obsaženo v Příloze A na straně 47.

2.4 Pracovní zařazení

Do firmy jsem nastoupil na pozici automatického testera. Mým úkolem bylo vytvořit kompletní automatickou obsluhu aplikace myTEAM®. Pro vytvoření takové obsluhy bylo důležité pochopit hlavní principy celé aplikace a připravit spolehlivé řešení, které pomůže při testování manuálním testerům. Byl jsem členem vývojového týmu, který celou aplikaci vytváří a udržuje.

Kapitola 3

Seznam zadaných úkolů v průběhu odborné praxe

Během mé odborné praxe ve firmě Kvados jsem řešil jeden hlavní úkol, kterým bylo vytvořit automatickou obsluhu aplikace myTEAM®. Z tohoto hlavního úkolu poté vznikaly menší podúkoly. Na začátku práce nebylo možné vyjmenovat přesně zadané menší úkoly, ty začaly krystalizovat až při vývoji. Důležité také bylo určení priorit a směr, jakým se má tento nástroj k automatickému testování ubírat. V tomto ohledu mi vždy pomohl můj konzultant Antonín Vaněček, který se mnou vše rozebral a domluvili jsme se na další postupu.

Prvním větším úkolem bylo úplné seznámení s celou aplikací, která je velice obsáhlá. Ve firmě jsem absolvoval několik porad a školení, kde mi bylo podrobně vysvětleno, jak daná aplikace funguje. V rámci času mezi poradami jsem měl možnost si aplikaci vyzkoušet i sám. Tento čas byl pro mě velice důležitý pro budoucí rozvoj testovacího nástroje. Celý tento proces zaučování mi zabral zhruba dva týdny.

Během dalšího týdne jsem vyzkoušel, jestli lze použít standardní nástroj od Selenia bez nějakých větších úprav, popřípadně nějaký jiný již existující. Konečné rozhodnutí bylo, že se vytvoří vlastní framework, který bude postaven na základech a principech již existujícího Selenia. Tato technologie byla zvolena po několika diskuzích v rámci vývojového týmu, při kterých jsme si ujasnili, co má tento nástroj v budoucnu vše umět.

Vlastní framework byl zvolen hlavně z důvodu, aby v něm bylo možné vše přizpůsobit aplikaci myTEAM®, která se v mnoha věcech liší od jiných aplikací. Dalším důvodem bylo, že měl tento testovací nástroj z malé částí testovat i jiné věci, než jen uživatelské rozhraní. Jedná se například o testování dat v samotné databázi nebo složité nastavování a testování práv. Ze začátku jsem měl velmi málo zkušeností s automatickým testováním, takže rozhodnutí padlo hlavně na základě zkušenějších kolegů a potřeb celé firmy.

Poté jsem již začal s implementací vlastní obsluhy aplikace. Nejdříve jsem dostal za úkol vytvořit podporu pro testování různých typů panelů a velkého množství formulářových komponent. Ze

začátku nebylo potřeba mít pokryté všechny typy komponent, jelikož se některé v aplikace vyskytovaly pouze výjimečně. Základem bylo udělat podporu pro ty nejpoužívanější komponenty a panely. Tato práce mi zabrala zhruba tři týdny.

Všechny další hlavní úkoly, které jsem řešil v rámci této práce, jsou podrobně popsány v kapitole 9 na straně 30. Dále je tam také podrobně vysvětleno, jak se v jednotlivých úkolech postupovalo a co bylo potřeba vyřešit pro úplné splnění daného úkolu. Ke každému zadaní jsem obdržel také odhad časové náročnosti, který měl znázorňovat, jaký čas bych měl zadanému úkolu věnovat. Někdy tento odhad úplně neodpovídal realitě, jelikož jsem se u nějakého problému zdržel déle. Také se stávalo, že jsem se musel k již vyřešeným úkolům znovu vracet, jelikož někdy vzniklo nějaké řešení, které nebylo zcela správné nebo za nějaký čas už nebylo dále použitelné. Seznam úkolů je pro přehlednost zobrazen také v tabulce 3.1.

Úkol	Časová náročnost [dny]	Kapitola s popisem úkolu
Podpora klientských komponent	15	9.1.1
Scrolování na elementy v panelech	3	9.1.2
Hledání elementů	8	9.2
Čekání v testech	10	9.3
Synchronní akce	3	9.3.1
Asynchronní akce	3	9.3.2
Paralelizace	7	9.5
Práce s dokumenty	3	9.4
Testovací uživatel	2	9.7
Přidělování práv	4	9.7.1
Headless režim	5	9.6.1
Report testů	4	9.8
Testování po každé změně aplikace	3	9.9

Tabulka 3.1: Seznam hlavních úkolů řešených v rámci praxe

Kapitola 4

Manuální testování

Manuální testování provádí lidé, kteří musí ručně otestovat předem zadaný úkol. Pokud se tento typ testování podíváme v širším kontextu, tak je zřejmé, že je možné takto otestovat většinu zadaných úkolů. Ať už se jedná například o správnost analýzy, která musí dávat smysl a nesmí obsahovat nesrovnalosti nebo nejasné situace. Dále se dá ověřit samotné nasazení nové verze aplikace nebo její vizuální vzhled. Možností pro toto testování je opravdu mnoho. Důležité je pak vybrat to, kde se stávají nejčastější chyby a také kde se náklady nevyšplhají do závratných čísel.

Z hlediska UI se testování provádí například podle zadaných scénářů. Tyto scénáře v ideálním případě připravují analytici, kteří spolu s analýzou sestaví sadu kombinací, která se může v dané části objevit. Jelikož analytik sepisuje chování určitého procesu, neměl by být pro něho problém také sestavit tyto scénáře pro testování. Je ale také možné, že analytik pouze předá analýzu jinému člověku, který z ní sestaví testovací metodu. V menších firmách se také stává, že se vůbec testovací scénáře nevytvářejí a vše záleží pak pouze na testerovi.

Toto testování se provádí často na místech, kde je automatizace velmi finančně nákladná nebo není vůbec možná. V praxi má ale toto testování stále velké zastoupení, a to i přesto, že se stále více firem snaží co nejvíce procesů zautomatizovat. Některé firmy můžou odradit častokrát větší náklady do zavedení nějakého typu automatizované obsluhy pro testování.

4.1 Výhody

Jelikož veškeré testování provádí člověk, může vidět celou řadu spojitostí, kterou naprogramovaný software nevidí. Díky svých zkušeností dokáže tester přijít na chybu rychleji a odhalit tak potenciální problém. V případě změny v aplikaci se přepíše testovací scénář a tester pak tuto část otestuje dle nového scénáře, popřípadně pokud takový scénář není k dispozici, pouze otestuje danou část dle zadaných pokynů. Takové testování lze provádět také jako součást vývoje a funkčnost testovat pravidelně a opakovaně.

4.2 Nevýhody

Největší nevýhodou manuálního testování je, že i v případě nepatrné změny je potřeba otestovat celou aplikaci úplně od začátku. Takové testování bývá sice účinné, ale je velice zdlouhavé a finančně náročné. V praxi pak na něho nezbývá moc času a otestuje se pouze upravená část aplikace.

Firmy jsou často stresovány termínem odevzdání práce zákazníkovi, který pokud nedokáží splnit, vystavují se riziku sankcí. Spoustu věcí se vyvine v tom nejposlednějším termínu a testeři musí za minimální čas vše důkladně otestovat.

Nevýhodou také může být, že při každém testování je potřeba brát v úvahu veškeré výjimky a zvláštní věci, které se v aplikaci nachází. V automatickém testu na takovou věc stačí myslet pouze při prvním napsání testu a při dalším testování na ni není potřeba myslet.

4.3 Využití v praxi

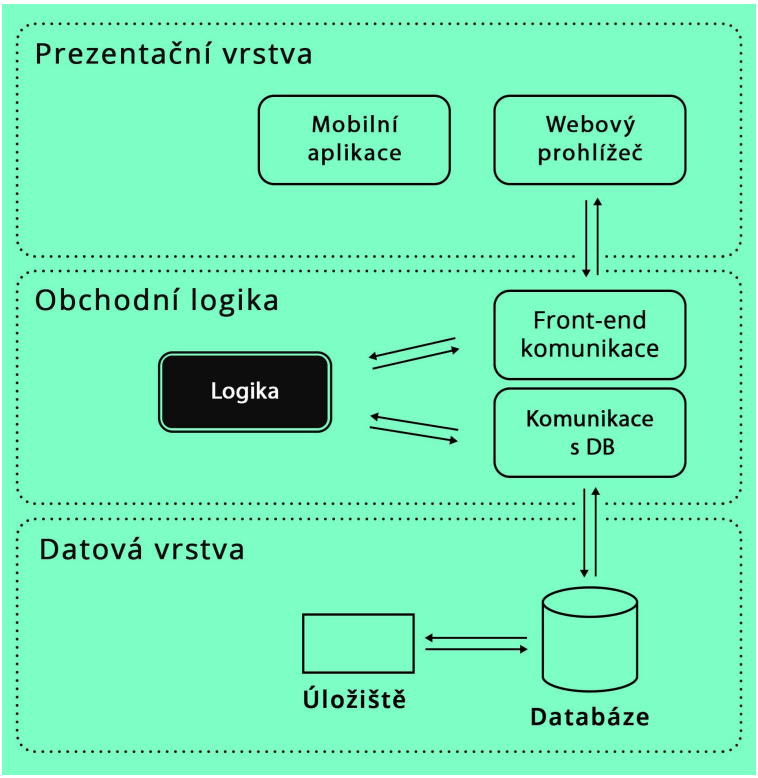
Tento typ testování se v praxi ve většině firem nadále praktikuje, i když stále častěji se takové týmy zmenšují a část těchto kapacit se přesouvá k automatickému testování. Manuální testování má ale nadále velké zastoupení a ještě nejspíše nějakou dobu mít bude. Samotný uživatel, lépe řečeno člověk, je totiž schopen odhalit i jiné věci, než jen ty, které jsou uvedené v testovacím scénáři. Celkově dokáže o těchto věcech přemýšlet v různých kontextech a je schopen se přizpůsobit různým změnám v aplikaci, které v rámci vývoje často nastávají.

Kapitola 5

Automatické testování

Automatické testování je dnes velice využívané a neustále jeho popularita roste. V posledních letech do automatizace proudí velké finanční prostředky a ani testování není výjimkou. Aby se mohla aplikace takto otestovat, tak je potřeba mít vhodný nástroj, který bude provádět obsluhu dané aplikace a jasně definované vstupní a výstupní data.

Automatické testy můžeme jednoduše rozdělit do třech kategorií, znázorněné jsou na obrázku 5.1.



Obrázek 5.1: Příklad rozdělení aplikace na její jednotlivé části[9]

Jako první je prezenční vrstva. Jedná se zde hlavně o UI testy, které se používají, pokud je nutná velmi častá interakce s uživatelem. Jako další lze testovat obchodní logiku, která se testuje například pomocí Unit testů. Důležitou součástí je také vrstva datová, ze které aplikace čerpají většinu svých informací, které se následně zobrazí uživateli.

5.1 Výhody

Největší výhodou u automatického testování je jednoznačně jejich znovu použitelnost. Pokud je správně napsán automatický test, je možné ho spouštět stále opakovaně, což ušetří nejen spoustu času, ale také financí.

Po uzavření nové verze je zpravidla nutné tuto aplikaci znovu celou otestovat. Pokud bychom neměli žádný automatický test a naše aplikace by byla rozsáhlá, tak je velká pravděpodobnost, že nedokážeme v rozumném čase přetestovat všechny možné kombinace, které by mohly nastat. Reálně pak hrozí, že by v aplikaci mohla zůstat nějaká kritická chyba.

Automatický test stačí napsat pouze jednou a v ideálním případě už takový test není potřeba nikdy měnit. Je tedy dobré psát takové testy již na ucelené části aplikace, které nebudou procházet rozsáhlým vývojem.

5.2 Nevýhody

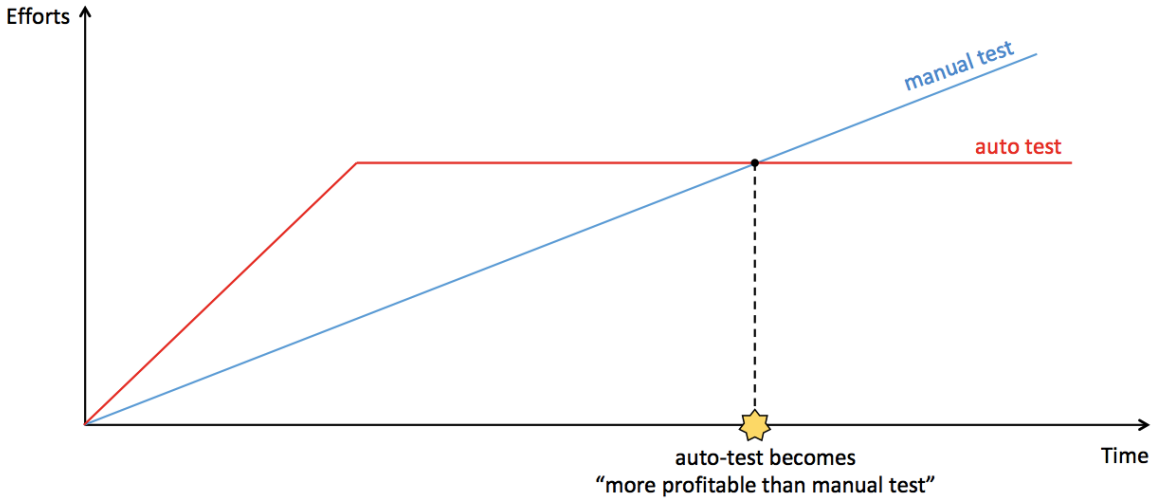
V případě větších aplikací bývá často problém, že aplikace obsahuje nějaké speciality, které nelze úplně jednoduše otestovat. V takovém případě se například u UI testů často vyvíjí vlastní framework, který sice na začátku stojí firmu dost času a financí, ale později se ji tato investice začne vracet.

V případě změny v aplikaci je potřeba udržovat aktuální také automatické testy. To samozřejmě zvětšuje časovou náročnost. Pokud budou vznikat testy na nedokončené části aplikace, tak reálně hrozí, že po nějaké změně bude nutné tyto testy opravit nebo v nejhorším celé přepsat.

Obecně pak bývá problém nalézt správně vyváženou křivku, kdy jsou automatické testy přínosem a co vše do nich ještě zahrnout a naopak co již nikoliv. Pokud by vznikalo velké množství testů a jedno otestování celé aplikace by zabralo větší časový úsek, může se stát, že toto testování bude kontraproduktivní.

Pro názornou ukázku slouží graf na obrázku 5.2, na kterém lze vidět, že se v určitém momentu setká manuální testování s automatickým. Při menším testování jednoznačně vyhrává to manuální. Pokud je ale potřeba testů více, pak je v grafu hezky vidět, že tam už vede testování automatické.

Reálně to tedy znamená, že na začátku bude investice do automatických testů větší, postupem času se ale toto řešení ukážu jako výhodnější.



Obrázek 5.2: Srovnání automatického a manuálního testování[10]

5.3 Využití v praxi

V ideálním případě chceme po každé změně celou aplikaci znovu přetestovat. U některého typu testování to nebývá problém a testy se můžou spustit přímo po sestavení kódu. Všechny typy testů ale nelze spustit po každé změně, takže se často využívá, že se aplikace testuje například jednou denně v předem stanovený čas, zpravidla někdy v noci.

Dále se aplikace testuje například před uzavřením verze. Někde bývá pravidlem, že pokud běh testů neskončil úspěšně, tak nelze ani uzavřít novou verzi aplikace, která je určena pro distribuci přímo k zákazníkovi.

Kapitola 6

UI testování

Testování uživatelského rozhraní je čím dál složitější, protože se do aplikací přidává mnoho různých speciálních efektů od různých animací, reklam nebo propracované grafiky. Obecně jde vidět, že jsou aplikace mnohem pestřejší a je kladen velký důraz na samotný vzhled.

Do tohoto typu testování patří například ověření správné pozice obrázků a jiných grafických prvků nebo zobrazení správných dat. Taková data se mohou měnit při nějaké změně. Typickým příkladem bývá přidání nového prvku nebo editace již existujícího. Po provedení této operace je znovu nutné zkontrolovat, jestli se vše zobrazuje uživateli správně. Určité změny mohou být doprovázeny také notifikacemi, které je nutné také zkontrolovat, jestli se v aplikaci zobrazují správně.

Při automatickém UI testování lze otestovat také více vrstev, než je tu prezenční. Obecně se to moc nedoporučuje, ale pokud aplikace jiné testy nemá a ani nemá v plánu do jiných investovat, i toto lze brát jako přívětivé řešení. Díky toho bude možné ověřit například nejen to, že se prvek správně zobrazil uživateli, ale také je možné zkontrolovat jestli se vše správně aplikovalo do datové vrstvy.

6.1 Automatické UI testování

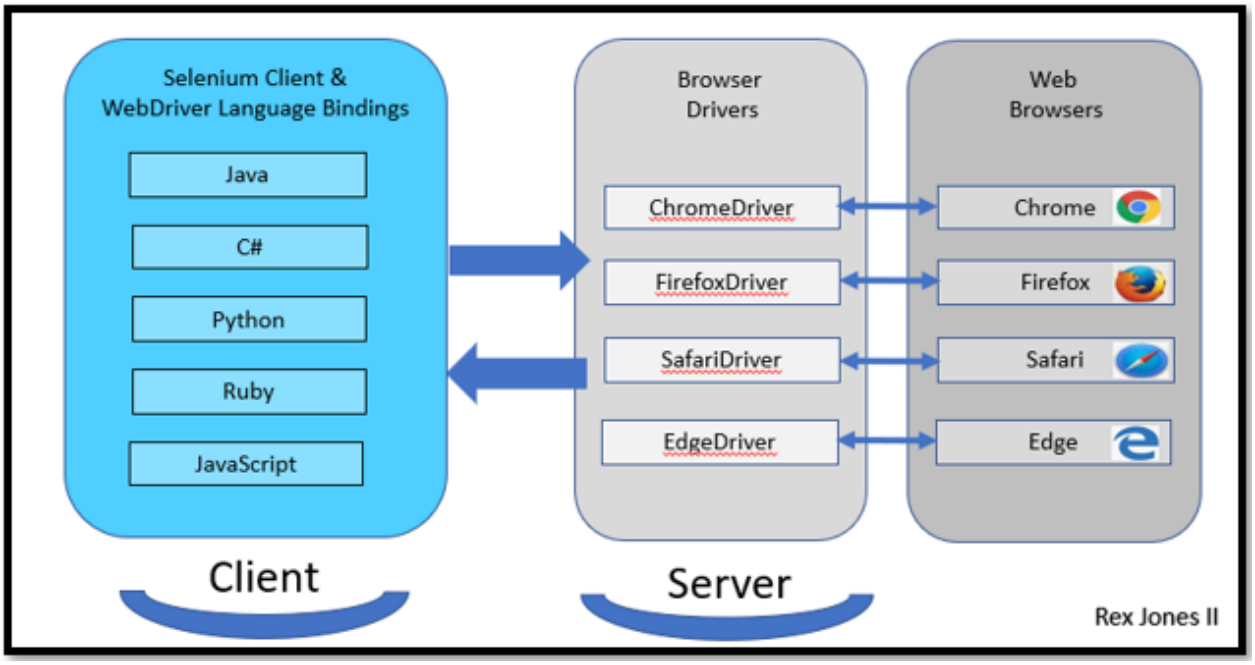
Základním principem v automatickém UI testování je naučit testovací metodu, framework nebo testovací nástroj, chování reálného uživatele. Je důležité vyzkoušet všechny možné scénáře a situace, které se v aplikací nacházejí. Je nutné počítat s tím, že libovolná změna může aplikaci kdykoliv pokazit, například když v aplikaci vyskočí modální okno, přeruší se spojení s internetem, zobrazí se nějaká validace nebo aplikace spadne na neošetřenou chybu.

V praxi se používají dva hlavní typy testovacích softwarů. První typ je založen na tom, že programátor nebo tester píše test pomocí kódu. To většinou vyžaduje určitou znalost programování. Výhodou pak bezesporu je, že nejste omezení jen daným softwarem, protože si můžete naprogramovat širokou škálu vlastních věcí, které budete potřebovat. Mezi takové testovací programy patří Selenium.

Druhý typ testování je mnohem jednodušší a nevyžaduje žádnou znalost základů programování. Testerovi stačí znát jen hlavní principy dané oblasti v aplikaci, na kterou má vytvořit samotný test. V takových nástrojích jde velká část objektů nalézt pouhým kliknutím na místo, kde se tento objekt zrovna nachází. Pomocí připraveného testovacího nástroje pak může jednoduše doplnit libovolnou podmínku, například na obsah určitého textu nebo viditelnost nějakého prvku. Po vytvoření takového testu je možné ho opakovaně spouštět bez další časové investice. Takové testování je vždy limitováno použitým nástrojem. Hodí se většinou pro menší projekty nebo pro aplikace, které používají standardní řešení a nemají žádné další vlastní požadavky. Velmi používaným programem je zde například TestComplete.

6.2 Selenium

Selenium bylo vymyšleno v roce 2004 Jasonem Hugginsenem a slouží jako nástroj pro testování webových aplikací. Můžeme ho zařadit do kategorie open source, je tedy zcela zdarma. Tento software podporuje mnoho programovacích jazyků, jako jsou například Java, Python, JavaScript nebo C#. Selenium podporuje také velké množství internetových prohlížečů - Chrome, Edge nebo Firefox. [11]



Obrázek 6.1: Zjednodušená architektura Selenia[12]

6.3 TestComplete

TestComplete je automatizovaná testovací platforma vyvinutá společností SmartBear Software v roce 1999. Patří mezi placené programy a poskytuje širokou škálu možností testování. Lze vytvářet testy pro webové a desktopové aplikace nebo pro mobilní zařízení, kde je kompatibilní s platformami Android i iOS. Podporuje také mnoho jazyků, například Javascript, Python nebo C#. Samotné testy se dají vytvářet jednoduchým klikáním na jednotlivé elementy. Těm je pak potřeba definovat, co se má na nich ověřit. Například je možné označit element typu text a ověřit jeho konkrétní hodnotu. [13]

Jedná se celkově o velice kvalitní nástroj, který ale narozdíl od Selenia je placený. Pořízení takové licence pak není vůbec levná záležitost. Cena se odvíjí od toho, jestli budeme používat aplikaci, která běží na počítači, mobilu nebo webu. Základní cena startuje na pěti tisících euro. [14]

Obrázek 6.2: Úvod do TestComplete[15]

Kapitola 7

Vytvoření vlastního frameworku postaveném na Seleniu

Návrh vlastního frameworku je vcelku složitý. Je potřeba si uceleně pojmenovat a nejlépe také sepsat všechny věci, které musí takový nástroj splňovat. U menších aplikací se často dojde k názoru, že investice do takové vlastní nadstavby je zbytečná a lze použít nějaké jiné dostupné řešení. Pokud je ale přesto takový framework potřeba, je dobré začít testováním jednotlivých komponent. Po otestování všech komponent samostatně lze pak jednoduše vytvářet složitější testovací scénáře pouhým skládáním těchto komponent do jednoho testu.

7.1 Inicializace dat

Než vůbec začneme psát první test, tak je potřeba si pro něho připravit základní data. V mnoha aplikacích je potřeba se do aplikace přihlásit, což obnáší založení takového uživatele s heslem. Všechny data je potřeba mít připravené tak, aby se dala v aplikaci používat opakovaně. K přípravě dat bychom měli využít mock. Mock je nasimulovaný objekt, který imituje pravou instanci třídy. V podstatě nahrazuje reálný objekt testovací fasádou a na nahrazovaném objektu neprovádí žádnou funkcionalitu. Mezi jeho největší výhody patří rychlost a pohodlnost. Díky toho není potřeba složitě otevírat spojení s databází a zakládat tam konkrétní data. Většinou se využívají tři různé stupně pro mockování dat:

- Data inicializované pro všechny testy
- Data inicializované pro modul/agendu
- Data inicializované pro konkrétní automatický test

V tomto frameworku se ale data zakládají přímo do databáze, jelikož mockování počítá s tím, že se testuje pouze doménová vrstva, která předpokládá, že vše ostatní funguje v pořádku. Zde se ale testuje také vrstva datová, takže mockování nebylo možné použít.

7.2 Přidělení práv

Přidělování a testování práv je u velkých aplikací velice problematické. Většina robusních systémů má velký a složitý systém práv. Pro automatické testování je důležité, aby měl vždy vytvořený uživatel přidělené jen ta práva, které nutně potřebuje pro vykonání daného testu. Díky toho bude schopný otestovat určenou funkcionalitu a zároveň tím můžeme ověřit správnou funkčnost práv. Je dobré mít také test, který otestuje aplikaci s uživatelem, který nemá přidělené žádná oprávnění a může vykonávat pouze akce, na které práva nepotřebuje.

Pokud budeme mít testovací metodu na založení nějakého záznamu, tak testovací uživatel by měl mít právo jen na toto založení. Proto je dobré v tomto testu také ověřit, jestli tento záznam nepůjde editovat, jelikož uživatel na editaci nemá přidělené žádné právo.

V praxi může existovat složitější systém práv, než jen práva pro konkrétního uživatele. Například práva pro určitou firmu, organizační jednotku nebo roli. Princip testování je stejný, jen je potřeba provést takový test na různé kombinace uživatelů.

V aplikaci myTEAM® je systém práv velmi robusní a při testování je na něj kladen velký důraz. V aplikaci se vyskytují různé dokumenty, ve kterých se nachází důvěrné informace o domluvených zakázkách a konkrétních cenách.

Před každým testem se tedy založí uživatel, kterému se nastaví také daná práva. Ty se po zápisu do databáze musí ještě propsat na více míst, což chvíli trvá. Proto v jádře testů existuje způsob, který čeká na přegeneraci těchto práv. Pokud se tato přegenerace nedokončí, nebude spuštěný ani samotný test.

7.3 Kontrola dat v datové vrstvě

UI testy mohou sloužit i pro testování datové vrstvy, i když k tomu primárně nejsou určené. Před otestováním prvků v aplikaci můžeme ověřit, jestli se nějaká data správně zapsala do databáze. Pokud ano, můžeme pokračovat na otestování přímo v aplikaci. Pokud nemáme k dispozici jiné typy testů, je dobré tuto kontrolu provádět aspoň přímo v UI testech.

Pro jednodušší operace typu kontroly založení, editace a smazání záznamu v tomto vlastním frameworku budeme používat tuto kontrolu datové vrstvy. Pro složitější operace, kdy je potřeba ověřovat velké množství dat nebo více tabulek již toto řešit nebudeme.

7.4 Kontrola v UI

Hlavní funkcí je pak samozřejmě kontrola přímo v aplikaci. Můžeme kontrolovat prakticky vše, co je potřeba. Například obsah číselného nebo textového elementu, porovnávat obrázky, grafy nebo pozici daných elementů.

V jazyce C# se používá pro kontrolu třída Assert. Ta nabízí velké množství metod. Pomocí této třídy lze ověřit například již výše zmíněný obsah nějakého elementu. Pokud by se takový obsah neshodoval s požadovaným výsledkem, pomocí této třídy lze jednoduše vypsat chybovou hlášku.

Často krát se také UI testy používají pro kontrolu nějakých procesů. Například vytvoření nějaké položky objednávky a zaplacení dané objednávky. Reálně lze vytvořit test na spousty různých kombinací. Je ale otázkou, jestli bude nějaký složitý test efektivní. Je potřeba zvážit, kolik zabere času vytvoření takového testu a kolik následně může ušetřit času. Ne vždy může být takový UI test efektivní. Pro testování procesů je častokrát vhodnější použít například unit test, který ověří celý proces bez nutnosti interakce s uživatelem, neboli bez uživatelského rozhraní.

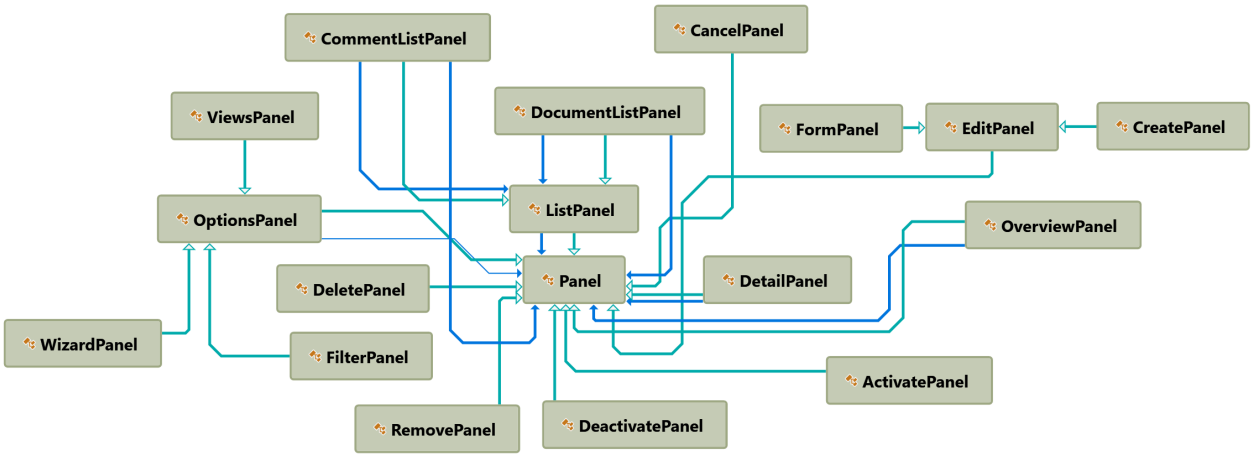
Kapitola 8

Hlavní třídy vlastního frameworku

V jádru projektu se nachází dvě hlavní třídy - Element a Panel. Obě třídy slouží pro práci s konkrétními prvky v uživatelském rozhraní aplikace myTEAM®. Většina dalších tříd bude dědit právě z těchto dvou tříd, které obsahují nejdůležitější část implementace.

8.1 Panely

Základním stavebním kamenem aplikace myTEAM® jsou její panely. Ve frameworku pro automatické testy to platí také. I zde se vše točí kolem panelů. Právě díky nich se tato aplikace odlišuje od ostatních. Každý panel je samostatný a jsou na něm zobrazená jedinečná data, která se ze serveru načítají asynchronně.



Obrázek 8.1: Diagram tříd pro práci s panely

V aplikaci existuje několik typů panelů. Aby bylo jednodušší s těmito panely při psaní testu pracovat, tak každý typ panelu má svoji třídu. To je důležité také pro budoucí rozvoj, kdy by se

mohla velká část kódu generovat z metadat. Na těchto třídách existují metody, které obsluhují určité akce, které se na daném panelu nachází.

Například panel dokumentů obsahuje metody, které umí nahrát dokument, otevřít složku nebo zobrazit hlavní složku ve stromové struktuře dokumentů.

8.2 Elementy

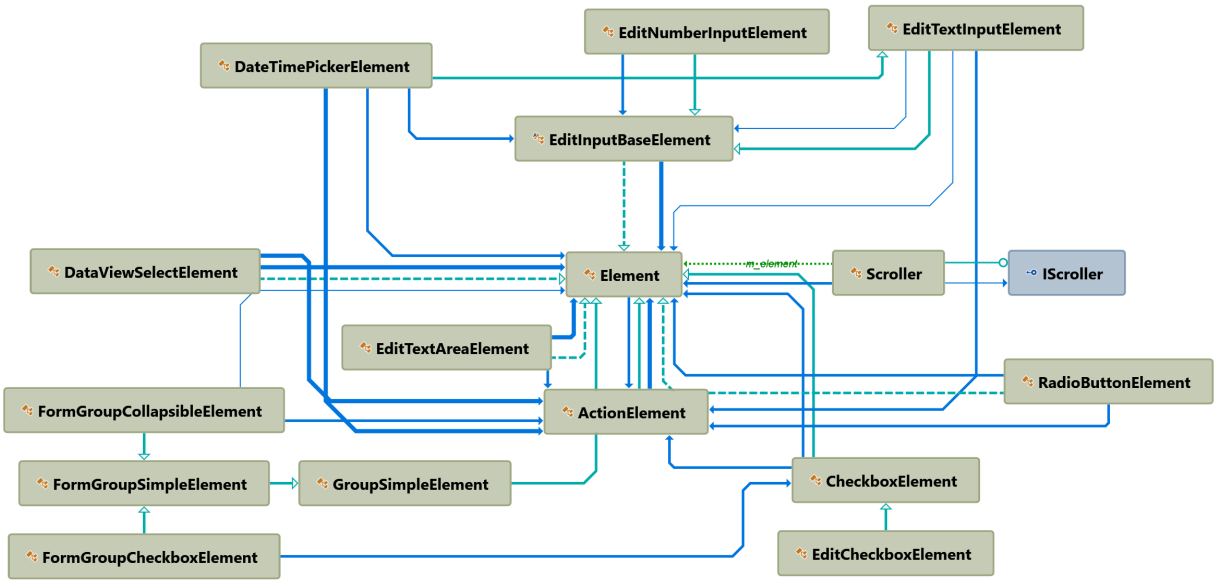
Jako element jsou nazvané všechny prvky, které lze v aplikaci nějak obsluhovat nebo s nimi provádět nějaké operace. Stejně jako u panelů má zde každý element svoji vlastní třídu. Díky toho je možné specifikovat název tagu v html. Hledání elementu bude probíhat nejdříve podle klíče. Pokud takový klíč byl nalezen, zkontroluje se také zvolený tag. Pokud se tento tag nebude shodovat, test skončí neúspěchem a vypíše tuto specifickou chybu. Tato kontrola zamezí mapování komponent na jinou třídu než tu, která byla pro tento typ elementu vytvořena. Ukázka kódu je dostupná zde 8.1.

```
public void CheckElementIsMappedToCorrectComponent()
{
    string expectedTagName = m_expectedTagName;
    string expectedClass = m_expectedClass;

    if (expectedTagName == null && expectedClass == null)
        return;

    string actualTagName = WebElement.TagName ?? string.Empty;
    string actualClass = WebElement.GetAttribute("class") ?? string.Empty;

    if (expectedTagName != null)
    {
        if (string.Equals(expectedTagName, actualTagName, StringComparison.
            CurrentCultureIgnoreCase))
        {
            return;
        }
        throw new Exception($"The element by '{FoundBy}' had tagName '{
            actualTagName}', the '{GetType().Name}' has to have a tagName '{
            expectedTagName}'.");
    }

    if (expectedClass != null)
    {
```

```
        if (actualClass.IndexOf(expectedClass, StringComparison.
            CurrentCultureIgnoreCase) >= 0)
        {
            return;
        }
        throw new Exception($"The element by '{FoundBy}' had class '{actualClass
            }', the '{GetType().Name}' has to have a class '{expectedClass}'.");
    }
}
```

Listing 8.1: Ukázka kódu na kontrolu mapování komponent ze třídy Element

Jako jeden z příkladů může být například vysouvací nabídka ve formulářích, v aplikaci pojmenovaná jako DataViewSelect. Pokud chceme po kliknutí na takovou nabídku vybrat nějaký prvek, který je v ní obsažen, tak tato operace bude implementována v její třídě, která bude dědit z hlavní třídy Element.



Obrázek 8.2: Diagram tříd pro práci s elementy

Kapitola 9

Hlavní úkoly při tvorbě vlastního frameworku

Aplikace myTEAM® je v mnoha ohledech specifická a odlišná od jiných aplikací. Pokud bychom použili pouze čisté Selenium bez vlastního frameworku, testy by nebyly spolehlivé. Při prováděných pokusech na začátku se použil pouze tento vytvořený framework, ale po pár testováních bylo jasné, že tudy cesta nepovede. Už na malém vzorku testů se testy chovaly nepředvídatelně a končily neúspěchem na náhodné chyby. Bylo tedy potřeba postupně ošetřovat jednotlivé problémové scénáře.

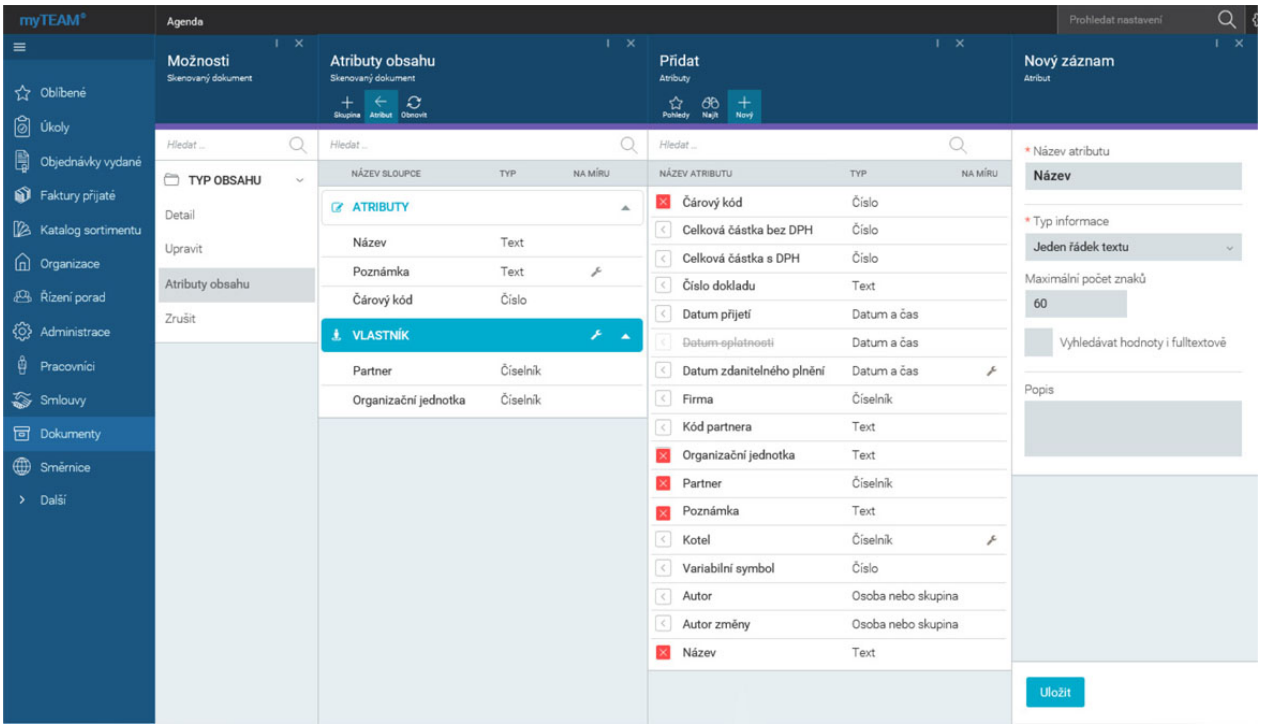
Pokud pomineme všechny problémy, díky kterých by testy nebyly spolehlivé, tak i přesto by bylo velmi obtížné psát větší množství nových testů. Proto vznikl vlastní framework, který slouží jako určitá nadstavba nad Seleniem. Ta má za úkol usnadnit psaní nových testů, řešit mnoho speciálních případů v aplikaci, provádět speciální kontroly a zabezpečit obsluhu celé aplikace.

9.1 Aplikace jako jednotlivé panely

Aplikace myTEAM® přichází na trh se zcela novou strategií jednotlivých nezávislých panelů, která se otevírají ve většině případů do pravé strany. Nezávislé je myšleno z toho hlediska, že veškerá data jsou v panelech načítaná asynchronně. To ale neznamená, že jeden panel nemůže ovlivnit jiný panel. Pokud budeme měnit záznam objednávky, tak je potřeba také zobrazit jiná data na panelu položky objednávky. Tyto panely můžou být zobrazeny současně, proto je potřeba umět reagovat na změnu daného záznamu. Tyto případy jsou zohledněny také v implementaci vlastního frameworku.

V testech je potřeba s panelem provádět různé operace, například ho minimalizovat nebo úplně zavřít. Po zavření panelu se také zkontroluje, jestli se panel opravdu zavřel. Dále se na panelech nachází dvě sady tlačítek - na vrchní a spodní části. Všechny tyto tlačítka umí framework obsluhovat pomocí svých metod.

Velmi důležité jsou potvrzovací tlačítka, která se nachází vždy ve spodní části. Pomocí nich se potvrzují všechny důležité akce, například založení, editace nebo smazání záznamů. Pokud by se záznamem nešla daná operace provést, zobrazí se chyba, kterou framework přečte a uloží ji do logu.



Obrázek 9.1: Ukázka aplikace myTEAM®[16]

9.1.1 Podpora klientských komponent

Jedním z mých prvních úkolů bylo vytvořit podporu pro testování různých typů panelů a velkého množství formulářových komponent, v tomto frameworku nazvané jako elementy. Na začátku nebylo potřeba mít implementované všechny typy těchto komponent a jejich kombinací. Některé se v aplikaci vyskytovaly pouze výjimečně. Základem bylo udělat podporu pro ty nejpoužívanější. Další implementace bude probíhat tehdy, pokud bude nutná pro konkrétní testovací metodu.

Komponenty mohou mít různý význam. Můžou sloužit pro zadání pouze číselných nebo textových hodnot, dále jako výběr více hodnot nebo nějaký přepínač mezi dvěma hodnotami. Podle potřeb každého elementu existuje třída, která s nimi provádí operace, které jsou v testovacích metodách potřeba.

9.1.2 Scrolování na elementy v panelech

Většina stránek používá jen jedno scrollování, které se nachází na pravé straně obrazovky a slouží k vertikálnímu pohybu. Není to ale pravidlem, nacházet se může nacházet také na spodní straně, pokud je potřeba provádět pohyby horizontálně.

V této aplikaci je scroller v každém panelu, který se nevejde na obrazovku. Pokud by bylo na obrazovce pouze standardní scrolování, bylo by možné využít již připravenou obsluhu, která je součástí Selenia. Takto bylo ale potřeba implementovat vlastní způsob scrolování.

Pokud existuje v aplikaci nějaký element, na který lze kliknout, dědí ze třídy ActionElement. Ta obsahuje právě metodu Click. Jedná se o velmi důležitou metodu, která musí zkontrolovat, jestli je takový element viditelný a jestli na něho vůbec lze kliknout.

```
public void Click(int offsetX = 0, int offsetY = 0)
{
    MakeClickable();
    try
    {
        new Actions(DriverInstance.WebDriver).MoveToElement(WebElement).
            MoveByOffset(offsetX, offsetY).Click().Build().Perform();
        Thread.Sleep(300);
    }
    catch (StaleElementReferenceException e)
    {
        throw new StaleElementReferenceException($"The element found by '{FoundBy
            }' threw a StaleElementReferenceException while performing a click
            Action.", e);
    }
}
```

Listing 9.1: Ukázka kódu ze třídy ActionElement

Problém nastává, pokud element není viditelný. To ještě ale neznamená, že na stránce, konkrétně v nějakém panelu, neexistuje. Na seznamu může existovat mnoho záznamů. Kdyby všechny záznamy a jejich údaje měly být zasílány ze serveru najednou, mohlo by to trvat i několik minut a není jasné, jestli by byl vůbec tento požadavek úspěšný. Proto se data načítají postupně po určitém počtu. To může způsobit, že záznam, který je potřeba najít, ještě nebude vůbec existovat, tudíž ani nebude viditelný. Zde přichází prostor pro scrolování, které je potřeba provádět až do té doby, dokud záznam nebude viditelný. Samozřejmě se může stát, že tam takový záznam vůbec neexistuje, i když se doscroluje až na konec seznamu. Potom jsou dvě možnosti. Může se scrolování zopakovat opačným směrem nebo vypsat chybovou hlášku a označit test jako neúspěšný. Pokud je element nalezen, vyvolá se akce samotného kliku a test pokračuje dále.

9.2 Hledání elementů

K hledání elementů se nejčastěji používá nějaký unikátní klíč, který musí být v rámci nějaké části jedinečný. Aplikace myTEAM® je rozdělena na jednotlivé panely, proto pokud je potřeba vyhledat nějaký element, je potřeba, aby tento element měl jedinečný klíč v rámci daného panelu. K tomuto jedinečnému klíči může být dále přidán také název tagu. Méně účinější by bylo hledání podle třídy

nebo nějakého kaskádového stylu. Takové hledání je ale vysoce neefektivní a mohlo by při něm docházet k chybám. Takový styl se může použít opakovaně a na jednom panelu by se mohl vyskytovat vícekrát. Nebylo by pak možné vyhodnotit, jaké hledání je to správné.

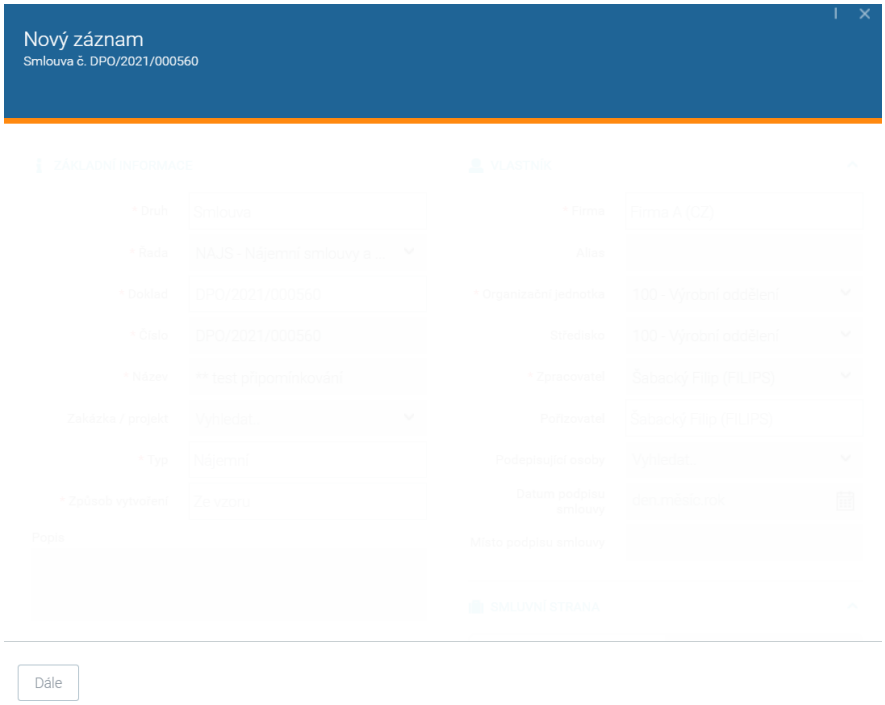
Tento jedinečný klíč byl ze začátku psaný do testů jako text. Později se přešlo na generování obsahu panelů do třídy, která obsahuje všechny názvy jednotlivých komponent. Díky toho lze jednoduše poznat, pokud byla nějaká komponenta odebrána nebo přejmenována. V tu chvíli se přegeneruje také tato třída a projekt s testy již nepůjde sestavit. Předtím docházelo k tomu, že test byl vyhodnocen jako neúspěšný až při hledání daného elementu přímo na stránce, což bylo velice neefektivní.

9.3 Čekání v testech

Pro aplikaci myTEAM® bylo potřeba vymyslet vlastní způsob čekání na dočtení každého panelu zvlášť. Po několika pokusech z toho vzešlo konečné řešení, které je funkční a stabilní. Každá akce, kterou bude testovací framework vykonávat, začíná kontrolou panelu. Na každém panelu může docházet ke dvou standardním případům, kdy se akce vykonává synchronně nebo asynchronně.

9.3.1 Synchronní akce

V případě synchronních akcí se zkontroluje, zda je panel již přístupný a není zablokován.



Obrázek 9.2: Synchronní akce v myTEAM® - zablokovaný panel

Pokud se provádí nějaká synchronní akce, například libovolné založení záznamu, uzavřou se všechny panely v aplikaci a nelze s nimi provádět jiné operace.

Tato situace nastává při složitějších procesech nebo akcích a může trvat i pár sekund. Panel se během toho překryje bílou částí a nelze na něm nic změnit nebo znovu kliknout na tlačítko. Pro kontrolu v testech se používá tento kód 9.2.

```
public void AssertLoaded(double? timeoutSeconds = null, double?
    initialWaitInSeconds = null)
{
    Log.LogMessage($"", DriverInstance);
    Stopwatch stopwatch = Stopwatch.StartNew();

    TimeSpan timeout = timeoutSeconds == null ? DriverInstance.ImplicitWaitTimeOut
         : TimeSpan.FromSeconds(timeoutSeconds.Value);
    double initialWaitForLoading = initialWaitInSeconds ?? DriverInstance.
        InitialWaitForLoading.TotalSeconds;

    GetElement(By.TagName("loading-overlay"), initialWaitForLoading);
    do
    {
        Element loadingGridItemElement = GetElement(By.TagName("grid-loading-item"
            ), 0);
        Element loadingOverlayElement = GetElement(By.TagName("loading-overlay"),
            0); //Can be zero, as we already waited above

        if (loadingOverlayElement == null && loadingGridItemElement == null)
            return;

        DriverInstance.CheckNoErrorModalText();

        Thread.Sleep(DriverInstance.SleepTimeout);
    } while (stopwatch.Elapsed < timeout);

    Assert.Fail($"Panel {FoundBy} didnt finish loading, waited {stopwatch.Elapsed}
        ");
}
```

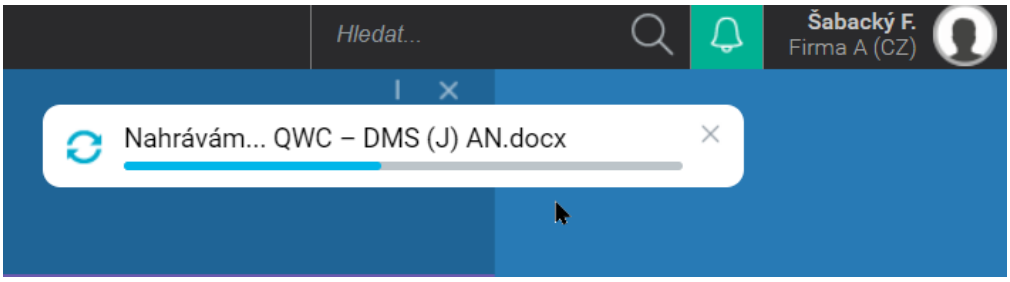
Listing 9.2: Ukázka kódu na kontrolu otevření panelu ze třídy Panel

9.3.2 Asynchronní akce

Jednou z nejtěžších věcí bylo vyřešit asynchronní načítání panelů a jejich dat. Většina standardních webových stránek při načítání dat ze serveru změní tlačítko "Načíst tuto stránku znovu"na jiný stav a obrázek. Díky toho lze jednoduše indikovat, jestli akce ještě probíhá nebo už byla dokončena. Toto čekání na dokončení akce umí rozeznat klasické Selenium. Asynchronní akce ale nelze takto jednoduše rozeznat.

Tato situace může nastat v aplikaci na libovolném seznamu. Při scrolování v něm se postupně dočítají další a další záznamy. Panel v tuto chvíli není nijak zablokován a lze s ním provádět i jiné akce, než jen procházet seznam.

Dalším případem mohou být náročnější operace, které nelze vykonávat synchronně. Mezi takové patří například nahrávání souboru s větší velikostí. Při takovém nahrávání je možné v aplikaci provádět také jiné akce jako třeba otevírat jiné panely nebo zakládat nové záznamy. Aktuální stav nahrávaného dokumentu zobrazuje notifikace, která se vždy nachází v pravém horním roku obrazovky.



Obrázek 9.3: Asynchronní akce v myTEAM® - nahrávání dokumentu

9.3.3 Vysvětlení principu čekání na dokončení akce

Pokud je potřeba počkat na dokončení libovolné akce, v tomto frameworku se použije vždy proces opakovaného provádění dané operace. V praxi to probíhá tak, že se automatický test snaží nalézt určitý záznam. Ten se mu na poprvé nepodaří z nějakého důvodu najít, například se ještě nestihl dočíst ze serveru. Akci nebude ihned opakovat znovu, ale použije příkaz Thread.Sleep, který slouží v C# k uspání aktuálního vlákna. Čas takového uspání musí být dobře vyladěný na většinu případů, aby se daná akce nemusela opakovat zbytečně vícekrát, ale naopak aby zbytečně daný test nebrzdila. Celý proces probíhá několikrát za sebou, například třicetkrát. Opakování není určeno konkrétními počty, ale časem. Vždy je určen nějaký maximální čas, po který se akce zkouší vykonávat znovu. Pokud se ani na poslední pokus nepovede nalézt hledaný záznam, test končí neúspěchem a stav testu se zapíše do logu. Více o logování je popsáno v kapitole 9.8 na straně 40.

Tento způsob implementace je aktuálně konečný a byl zvolen po delším rozvoji. Na úplném začátku byl použitý jednoduchý princip čekání v automatickém testu. Například ihned po začátku

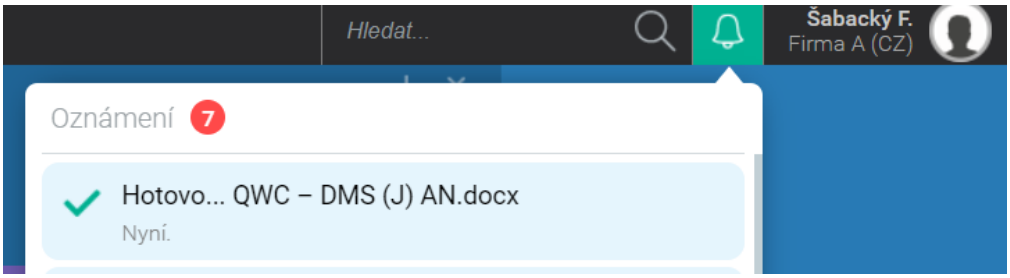
nahrávání souboru, se vlákno uspalo na předem zvolený čas - pět sekund. Do tohoto času se nahrání ve většině případů dokončilo. V určitých momentech se ale aplikace zpomalila a test byl neúspěšný. Docházelo zde k situacím, kdy se dokument nahrál někdy za dvě sekundy, ale někdy až za deset sekund.

Uspávání vlákna je potřeba používat minimálně a mělo by se vyskytovat pouze v bázových metodách, nikdy ne v konkrétním testu. Jeho chování je potřeba důkladně odladit napříč celou aplikací. Při použití v konkrétních testech by se mohlo jednat o neefektivní a nefunkční způsob, při kterém by test musel zbytečně čekat, i když byla akce již dokončena nebo naopak byl test neúspěšný, jelikož předešlý požadavek ještě nebyl dokončen, ale je předpoklad, že brzy dokončen bude.

9.4 Práce s dokumenty

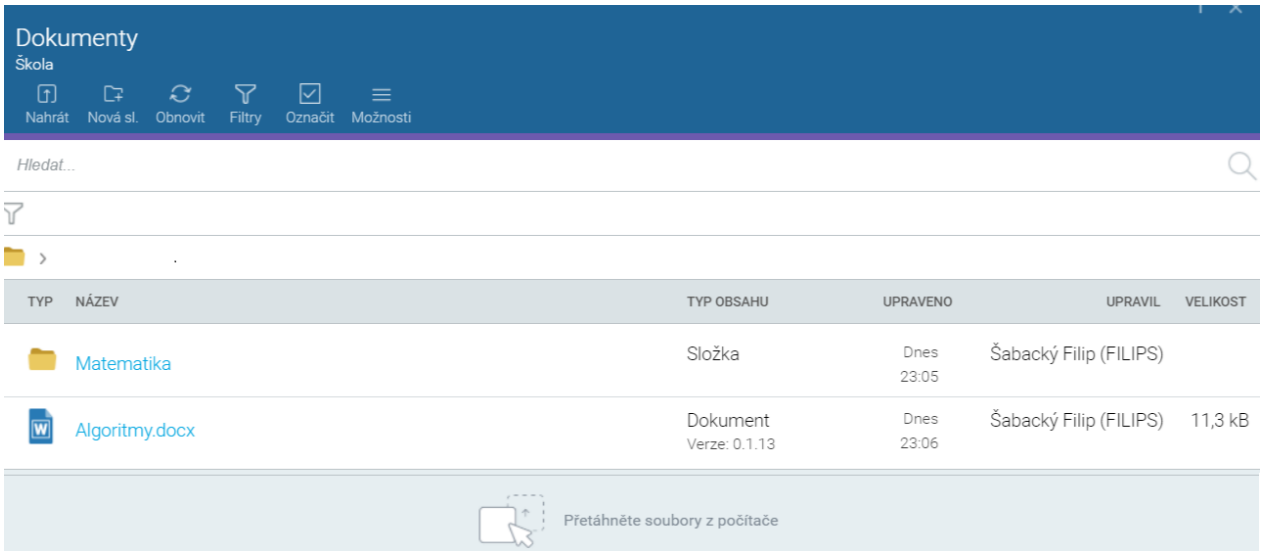
Hlavním modulem v aplikaci myTEAM® je bezesporu modul DMS. Proto bylo důležité pečlivě vyladit veškerou práci s dokumenty také při automatickém testování. Existují dva způsoby, kterými se může dokument do aplikace nahrát. Při prvních pokusech nahrávání nebylo spolehlivé a test končil náhodně neúspěchem. Dokument se ve většině případů sice nahrál, ale docházelo k náhodným situacím, kdy se dokument nahrát nepodařilo. Vše nakonec skončilo změnou implementace, která se již jeví spolehlivě.

Jakmile se dokument začne do aplikace nahrávat, zobrazí se notifikace, která indikuje stav nahrávaného dokumentu. Testovací framework nejdříve zkontroluje, jestli se tato notifikace vůbec zobrazila. Pokud ano, pokračuje dále a čeká, až tato notifikace úspěšně skončí. Taková notifikace je pak zobrazena na obrázku 9.4. V případě chyby se test ukončí a vypíše podrobný popis vzniklé chyby.



Obrázek 9.4: Úspěšné nahrání dokumentu v myTEAM®

První způsob, kterým lze dokument nahrát, je velice jednoduchý. Stačí pouze přetáhnout dokument z počítače na vyznačené místo v aplikaci. Takový přesun bylo nutné vykonat pomocí JavaScriptové akce. Ta má za úkol takový dokument vzít a na danou oblast opravdu přetáhnout. Ta se nachází ve spodní části panelu. Panel dokumentů je vidět na obrázku 9.5. Dále pak nahrávání postupuje stejným způsobem, s to tak, že vyskočí notifikace, dokument se nahraje a test pokračuje dále.



Obrázek 9.5: Panel dokumentů

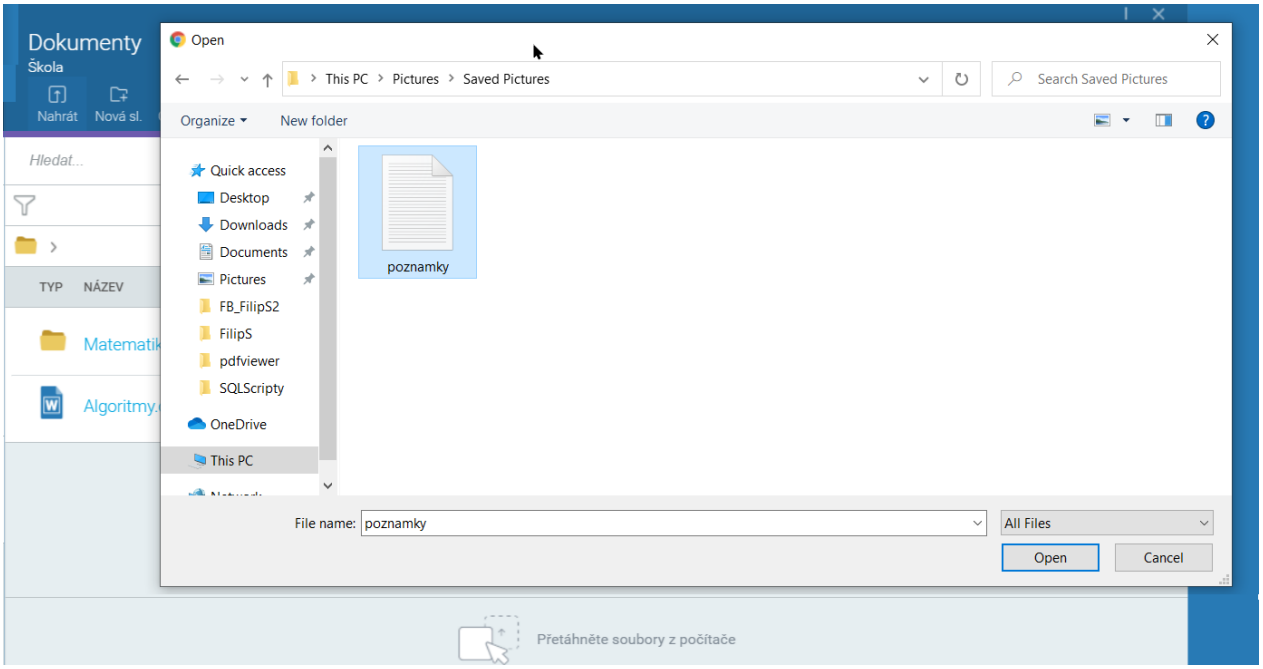
Důležité také je, aby se průběh uložil do logu. Pokud by nastala s nahráváním nějaká chyba, tak v ideálním případě při dobrém logování bude jednoduché vyčíst, která část operace se pokazila. Je také možné si během nahrávání udělat pár snímků z obrazovky. Může se jednat o místo, na kterém vzniká více chyb a snímky se při odhalování chyby můžou hodit. Na ukázce 9.3 je metoda, která se používá pro přetažení dokumentu na vyznačenou oblast v aplikaci.

```
public void DropFileOnElement(By by, string fileName)
{
    Log.LogMessage($"", this);

    IWebElement fileDropArea = WebDriver.FindElement(by);

    IJavaScriptExecutor jse = (IJavaScriptExecutor)WebDriver;

    string initEventJs = "files = [];"
                       + "files.push(new File([new Blob(['zkouska'], {type: 'text/
                           plain'})], '" + fileName + "'));"
                       + "var eve=document.createEvent(\"HTMLEvents\");"
                       + "eve.initEvent(\"drop\", true, true);"
                       + "eve.dataTransfer = {files:files};"
                       + "eve.preventDefault = function () {};"
                       + "eve.type = \"drop\";"
                       + "arguments[0].dispatchEvent(eve);";
```

```
    jse.ExecuteScript(initEventJs, fileDropArea);
}
```

Listing 9.3: Ukázka kódu ze třídy Driver

Při druhém způsobu stačí kliknout na tlačítko pro nahrání dokumentu. Následně se objeví standardní panel Windows, ve kterém si lze vybrat dokument.



Obrázek 9.6: Dokumenty - okno pro nahrání

9.5 Paralelizace

Jak testy začaly postupně přibývat, tak rostl přímou úměrou také čas dokončení všech testů. Cílem této práce není pokrýt celou aplikaci automatickými testy, ale připravit funkční řešení obsluhy aplikace. Proto bylo potřeba pár testů napsat a vyzkoušet jejich chování.

Už při jednotkách testů bylo jasné, že je potřeba tento framework připravit na to, aby mohlo běžet více testů současně. Když by v budoucnu vzniklo například jen dvacet testů a každý by trval v průměru dvě minuty, celý tento jeden běh by trval čtyřicet minut. Je ale předpoklad, že konečný počet testů se bude pohybovat v řádech několika stovek. Proto byla také paralelizaci věnována velká pozornost.

Již při prvních pokusech, kdy běžely pouze dva testy paralelně, se ale objevilo spoustu nových problémů, které vznikly díky chybným implementacím v tomto frameworku. Jednalo se například

o závislost nějakého testu na jiném testu. Například, test A počítal s tím, že se v aplikaci nachází záznam s nějakým jménem. Test B ale ještě předtím toto jméno změnil. To znamená, že i když test A byl napsaný správně z hlediska funkčnosti, tak byl neúspěšný.

Vše skončilo až u toho, že každý test, než vůbec začne, si do databáze založí všechny potřebná data, které potřebuje pro úspěšné vykonání testu. Pokud se jedná o nějaké názvy nebo jména, tak z důvodu kolizí jsou všechna tato data náhodně generována. Tato informace se skládá většinou z nějakého prefixu a části GUIDu, který nám díky jeho velkého množství kombinací zaručuje téměř jistou jedinečnost.

9.6 Velká náročnost na hardware

Pro běh testů nejsou potřeba žádné náročné hardwarové požadavky. Je potřeba, aby stanice zvládala plynule ovládání nějakého prohlížeče, ve kterém se bude testovat. Problém ale vzniká hlavně ve chvíli, kdy se testovací metody budou provádět paralelně. V tu chvíli bylo nutné tyto kapacity navýšit. Bylo nutné najít ideální počet testů, které mohou běžet současně na daném hardwaru, který firma již dále nechtěla zvětšovat. Ideálním poměrem se ukázalo využití čtyř vláken, tudíž běhu čtyř testů současně. Při menším počtu bylo jedno celé testování pomalejší. To samé se také ukázalo, pokud běželo více testů současně než čtyři.

9.6.1 Headless režim

Při testování je zatím použitý pouze prohlížeč Google Chrome. Při běhu více testů současně a nutnosti otevírat více instancí tohoto prohlížeče se generovaly obrovské nároky na procesor a vyrovnávací paměť. Jednou z možností bylo opět navýšit hardwarové kapacity, ale to nelze dělat donekonečna.

Skvělým řešením se ukázalo použití módu Chromu zvané jako headless. V tomto režimu prohlížeč normálně běží a podporuje všechny jeho funkce. Jediným rozdílem je, že jeho běh nelze vidět a tím pádem mnohonásobně zmenšuje požadavky na hardware.

Selenium tento režim plně podporuje. Jednoduše v něm provádí všechny potřebné operace a nedělá mu problém ani pořízení snímku obrazovky, nahrání dokumentu nebo další jiné funkce, které jsou poskytovány ve standardním režimu.

9.7 Testovací uživatel

Pro přístup do aplikace je potřeba mít vytvořený účet. Ten je sice možné založit přímo v aplikaci, ale pouze jiným uživatelem, který má na založení nového uživatele přidělené právo. Dalším způsobem je možnost si založit takový účet přímo do databáze. Takový způsob používají právě automatické testy. Před každým testem založí testovacího uživatele, pod kterým se tento test do aplikace přihlásí.

9.7.1 Přidělování práv

Pokud by se nově vytvořený uživatel do aplikace přihlásil, nebylo by mu to nic platné, protože by neměl přidělená žádná práva a neviděl by žádné moduly. Proto je nutné každému nově vytvořenému uživateli přidělit potřebná práva. Vždy je nutné přidělit jen ty nejnutnější oprávnění. Při testování je nutné se na toto zaměřit, jelikož jsou práva v aplikaci mohutně konfigurovatelná a velice složitá. Pokud je potřeba editovat libovolný záznam, přidělí se testovacímu uživateli pouze právo pro editaci, nikoliv na založení nového záznamu nebo odstranění již stávajícího.

9.8 Report testů

Pokud je test úspěšný, tak se většinou jedná o nedůležitou informaci, kterou není nutné zapisovat do logu. To ale neznamená, že to nelze. Jednou z možností je uložit důležité informace z míst, které jsou často chybové. V tomto frameworku toto využito není. Jediná informace, která se ukládá, je pouze zapsání všech testů, i těch úspěšných, na disk do souboru po jeho dokončení, ať už úspěšném nebo neúspěšném. Díky toho lze jednoduše poznat v rámci běhu všech testů, které testy již byly vykonané.

Ze začátku bylo velmi chybové místo při nahrávání dokumentů, takže se pořizovaly snímky z obrazovky a zápis informací před a v průběhu nahrávání. Nyní je ale tato funkčnost již bezchybná a není potřeba toto provádět.

Veškeré podrobnosti o konkrétním testu se zapisují na úložiště až poté, kdy test nebyl úspěšný. Zaznamená se akce, která se naposledy testu nepodařila provést, snímek obrazovky, důležitých informací obsažených v metodách a celý DOM stránky. Důležité je také uložení celého výpisu z konzole prohlížeče. Svoje logování má také server aplikace, ze kterého se také dají často zjistit konkrétní chyby. Čím lepší takové logování je, tím je jednodušší a rychlejší odhalit vzniklou chybu.

9.9 Testování po každé změně aplikace

Jelikož na aplikaci myTEAM® pracuje více programátorů, kteří nahrávají své změny na jedno hlavní místo, je potřeba po každé takové změně celou aplikaci znovu přetestovat. Vždy je riziko, že se nová verze aplikace mohla pokazit i na úplně jiném místě. Proto se používá regresní testování. Tato metoda nám slouží k tomu, že po provedené změně nebo implementaci nových vlastností nemělo vliv na jiné již stávající funkce, které se v aplikaci nacházejí.

Problém může nastat při vzniku velkého množství testů, kdy jeden běh všech těchto testů již bude trvat delší dobu. Pak se může stávat, že nebude možné provádět celé testování po každé změně od programátora. Bylo by nutné pak spouštět testy například na více strojích nebo provádět testování v předem daných intervalech, v krajním případě jen jednou denně.

Kapitola 10

Zhodnocení znalostí studenta pro vykonání odborné praxe

Bakalářské studium mě na tuto odbornou praxi velmi dobře teoreticky připravilo. Důležité pro mě byly nejen předměty se zaměřením na programování, ale také ty, ve kterých se rozvíjí nějakým způsobem logické myšlení. Při implementaci jsem využil mnoho znalostí z oblasti matematiky a teoretické informatiky.

10.1 Teoretické a praktické znalosti získané v průběhu studia

Nejdůležitější pro mě byla znalost programování v jazyce C#, ve kterém je celý framework vytvořený. Pokud bych tento jazyk vůbec neznal, tak bych nemohl ihned plynule začít s programováním této testovací obsluhy aplikace myTEAM®. Pomohli mi také předměty, které se zabývaly návrhem softwaru. Ujasnil jsem si, co opravdu ve vlastním frameworku potřebuji a jak správně celou architekturu navrhnout. Nevymyslel jsem ale vše sám, měl jsem kolem sebe tým lidí, který mi dokázal se vším poradit a nasměrovat mě správným směrem.

V průběhu studia jsem také získal dobré teoretické a praktické znalosti při tvorbě algoritmů. Na tuto problematiku se zaměřují dokonce dva předměty - Algoritmy I a Algoritmy II. Jejich obsah jsem při tvorbě této práce vnímal jako velmi důležitý.

10.2 Znalosti či dovednosti scházející studentovi v průběhu praxe

Přímo oblastí testování aplikací se bohužel žádný předmět v rámci bakalářského studia nezabýval. Veškeré informace jsem získal buď ve firmě KVADOS, a.s. nebo individuálním samostudiem z internetu. Tam existuje mnoho zdrojů, které se zabývají touto oblastí testování, zvlášť pak samotného UI. Určitě bych v rámci bakalářského studia podobný předmět uvítal.

10.3 Přínos odborné praxe

Celou odbornou praxi hodnotím velice pozitivně. Jsem nadšený a vděčný firmě KVADOS, a.s., že mi umožnila se zapojit do reálného projektu a že jsem mohl být také přínosem. Díky této odborné praxi jsem si mohl vyzkoušet aplikovat nasbírané teoretické znalosti v rámci studia přímo na reálném projektu v praxi. Naučil jsem se mnoho o testování a pochopil jsem, jak velký význam testování má na celý daný produkt.

10.4 Zkušenosti s novým frameworkem ve firmě

Při vývoji nového testovacího frameworku vznikalo také menší množství automatických testů, které vytvářeli jiní zaměstnanci. Díky toho, že tento nástroj používali, objevili v něm nějaké chyby a věci, které nebyly pro vytváření nových testů optimální. Po těchto zkušenostech bylo potřeba tyto připomínky zapracovat a nějaký kód přepsat tak, aby se lépe používal. Vždy byl kladen velký důraz na jednoduché a přehledné používání.

Napsané testy nyní firma používá jako první kontrolu, že je aplikace v pořádku a bez chyb. Testování probíhá po každé změně v aplikaci od libovolného programátora. V budoucnu se plánuje vytvoření automatických testů pro všechny agendy, které se nacházejí v aplikaci myTEAM®.

10.5 Testovací příručka

Během vývoje jsem vytvořil jednoduchou a krátkou testovací příručku pro potřeby firmy, ve které jsou popsané základní principy testování. Obsahem je také krátké seznámení s automatickým testovacím a celou touto novou technologií. Zaměřil jsem se také na to, abych zmínil důležité věci, kterým je potřeba se vyhnout nebo co je naopak správné použít.

Kapitola 11

Možnosti dalšího rozvoje

Tento framework samozřejmě nyní není v konečném stavu. Jak už to u každého softwaru bývá, vždy je možnost ho někde posunout, něco přepsat nebo něco vylepšit. Velmi důležité je stanovit nějaký plán rozvoje a jednoznačně určit priority. To ale neznamená, že by se v něm objevovaly nějaké chyby, které by bránily normálnímu používání.

11.1 Generovaní testovacích metod

Tím největším plánem je bezesporu generování testovacích metod. Mělo by se jednat o nějaký nástroj, ve kterém by se podle toho, na co manuální tester zrovna klikne, vygeneroval kód, který bude ve výsledku tvořit celou testovací metodu. Pokud by se tohle povedlo, bylo by velice jednoduché tvořit další nové testy.

S tímto použitím zatím nemám žádnou zkušenost, takže nedokážu ani zhruba odhadnout, jaká by byla časová náročnost něco takové vytvořit aspoň pro jednodušší použití.

Kapitola 12

Závěr

V rámci této práce se podařilo vytvořit automatizovanou obsluhu aplikace myTEAM®. Na začátku bylo důležité pochopit hlavní principy této modulární aplikace a daný nástroj na ni přizpůsobit. Díky vytvoření tohoto testovacího nástroje bude dále možné jednoduše pokrýt automatizovanými testy všechny moduly, například digitální evidenci dokumentů, smluv nebo objednávek.

Pochopil jsem, že je testování pro firmy velmi důležité a častokrát do něho investují mnoho finančních prostředků. V každém softwaru vzniká nespočet chyb a čím dříve se tyto chyby odhalí, tím to bude pro firmu lepší a hlavně levnější. Pokud se nepodaří chybu ani po důkladném manuálním a automatickém testování odhalit, tak hrozí, že se dostane až ke koncovému zákazníkovi. Pokud na ně zákazník narazí, tak v lepším případě kontaktuje zákaznickou podporu a požádá o opravu chyby. V horším případě se může stát, že se bude jednat o chybu kritickou, která zákazníkovi znemožní pokračování v práci a firmu poté neminou nemalé peněžité sankce.

Nejlepší variantou tedy je, aby se chyby k zákazníkovi vůbec nedostávaly. Z tohoto důvodu vznikl tento nástroj, který má sloužit k pravidelnému testování a má odhalit co nejvíce chyb za co nejmenší čas.

Celkově se také ukázalo, že byla správně zvolena technologie Selenia. Práce s ní byla jednoduchá, přehledná a nechyběla k ní ani přehledná dokumentace na webových stránkách.

Absolvováním bakalářské praxe jsem si mohl vyzkoušet v praxi teorii, kterou jsem načerpal během bakalářského studia. Zároveň jsem se naučil spoustu nových věcí nejen z oblasti automatického testování, ale také jsem začal chápat vývoj aplikací úplně jinak, než předtím. Uvědomil jsem si, jak moc je důležité kvalitní testování, kterému jsem v minulosti nepřikládal až tak velký význam.

Celkově pro mě byla praxe velkou zkušeností a získal jsem spousty nových znalostí při práci na reálných projektech. S výsledkem praxe jsem spokojený, ale samozřejmě je vždy co zlepšovat a doufám, že budu moct navázat na tuto práci také v budoucnu.

Literatura

1. *Testování UI* [online] [cit. 2021-03-10]. Dostupné z: https://sdtimes.com/wp-content/uploads/2020/02/UItesting-1-490x275.jpg.
2. *Kvados historie* [online] [cit. 2021-03-13]. Dostupné z: https://www.kvados.cz/o-spolecnosti/historie/.
3. *Kvados logo* [online] [cit. 2021-03-13]. Dostupné z: https://wwwold.kvados.cz/media/ke-stazeni/loga/logo_kvados_cmyk_large_backgroundblue.jpg.
4. *myCASH® POS* [online] [cit. 2021-03-13]. Dostupné z: https://www.kvados.cz/produkty/mycash/.
5. *mySTOCK® WMS* [online] [cit. 2021-03-13]. Dostupné z: https://mystock.kvados.cz/.
6. *myAVIS® CRM* [online] [cit. 2021-03-13]. Dostupné z: https://www.myavis.cz/.
7. *myVENTUS® ERP* [online] [cit. 2021-03-13]. Dostupné z: https://www.kvados.cz/produkty/ventus/.
8. *myTEAM®* [online] [cit. 2021-03-13]. Dostupné z: https://myteam.kvados.cz/.
9. *Příklad rozdělení aplikace na její jednotlivé části* [online] [cit. 2021-03-14]. Dostupné z: https://bonsai-development.cz/Content/appData/Articles/31/contentImages/rozvrstveni_aplikace.png.
10. *Srovnání automatické a manuálního testování* [online] [cit. 2021-03-28]. Dostupné z: https://miro.medium.com/max/2650/1*zxQN8qzglHl4-pd1GXGhWg.png.
11. YUJUN LIANG, Alex Collins. *Selenium WebDriver: From Foundations To Framework*. 2016.
12. *Zjednodušená architektura Selenia* [online] [cit. 2021-03-14]. Dostupné z: https://dzone.com/storage/temp/13113160-1-selenium-architecture.png.
13. ALPAEV, Gennadiy. *TestComplete Cookbook*. 2013.
14. *TestComplete ceny* [online] [cit. 2021-03-28]. Dostupné z: https://smartbear.com/product/testcomplete/pricing/.

15. *Úvod do TestComplete* [online] [cit. 2021-03-28]. Dostupné z: https://d24hrkfl5wrd0k.cloudfront.net/uploads/2020/07/BLOG_An-Introduction-To-%09%09%20Testcomplete-2-768x401.jpg.

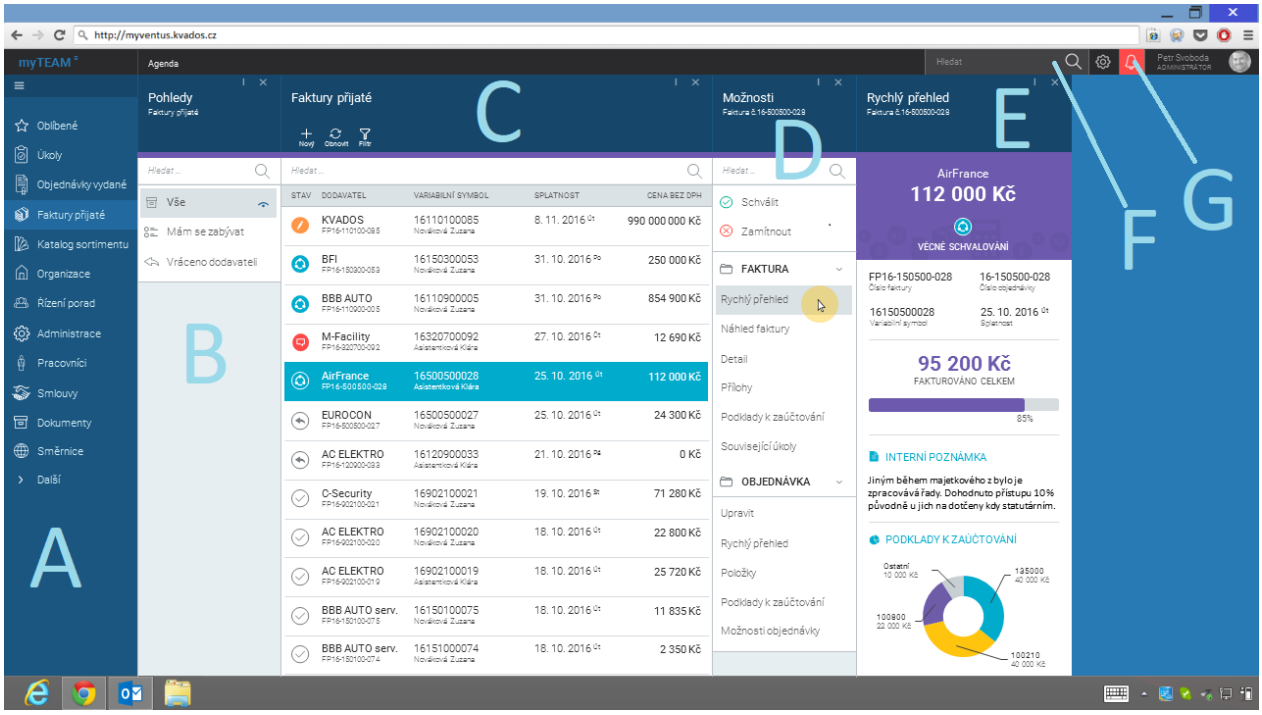
16. *Ukázka aplikace myTEAM®* [online] [cit. 2021-03-14]. Dostupné z: https://dzone.com/storage/temp/13113160-1-selenium-architecture.png.

17. *Ukázka modulu faktur aplikace myTEAM®* [online] [cit. 2021-04-25]. Dostupné z: https://www.digitalnicesta.cz/storage/solutions/sol0021__sit0021__rmdChFAP.png.

Příloha A

Představení aplikace myTEAM®

V práci byla testována hlavně prezenční vrstva aplikace, proto je zde krátký přehled toho, co aplikace myTEAM® nabízí z hlediska uživatelského rozhraní. Na obrázku A.1 jsou zobrazené panely, které se vztahují k modulu faktur. Na jednotlivých místech se nachází písmena, která slouží pro orientaci při popisu tohoto obrázku.



Obrázek A.1: Ukázka modulu faktur aplikace myTEAM®[17]

A - V levé části se nachází komponenta hlavní nabídky, která má pevně nastavené některé prvky, například modul Úkolů, Smlouv nebo Směrnic. Není pravidlem, že každý modul je jedna položka v hlavním menu, ale ve většině případů tomu tak je. Po najetí na některou z těchto

položek se může zobrazit vyjíždějící podnabídka, ve které je většinou možnost otevření panelu pro založení nového záznamu nebo jiná zrychlená volba. Za tyto statické položky lze přidat vlastní záznamové typy obsahu, které si uživatel vytvořil jako nějakou evidenci.

B - Panel pohledů úzce souvisí s filtrováním a má uživateli nabídnout předem definované filtry, které se v dané oblasti nejčastěji využívají. Další výhodou tohoto panelu také je, že si uživatel může vyfiltrovat záznamy v seznamu Faktur přijatých v panelu C a následně si toto filtrování uložit. Taková položka se pak bude zobrazovat právě v panelu pohledů.

C - Pro přehledné zobrazení základních dat slouží seznam, ve kterém lze zobrazit libovolné údaje. Vzhled tohoto seznamu je dán šablonou, kterou si může každý uživatel změnit nebo vytvořit úplně novou. Dále je také možné podle jednotlivých údajů data seřazovat, například podle stavu, splatnosti nebo ceny.

D - Panel možností je pro každý záznam jedinečný a liší se podle toho, jaké funkce má daná agenda nebo modul poskytovat. Jednotlivé položky panelu mohou být součástí nějaké skupiny. Na obrázku jsou dvě skupiny - Faktura a Objednávka. Ve většině případů je zde přístupná volba Rychlého přehledu, Detailu, Editace nebo Odstranění.

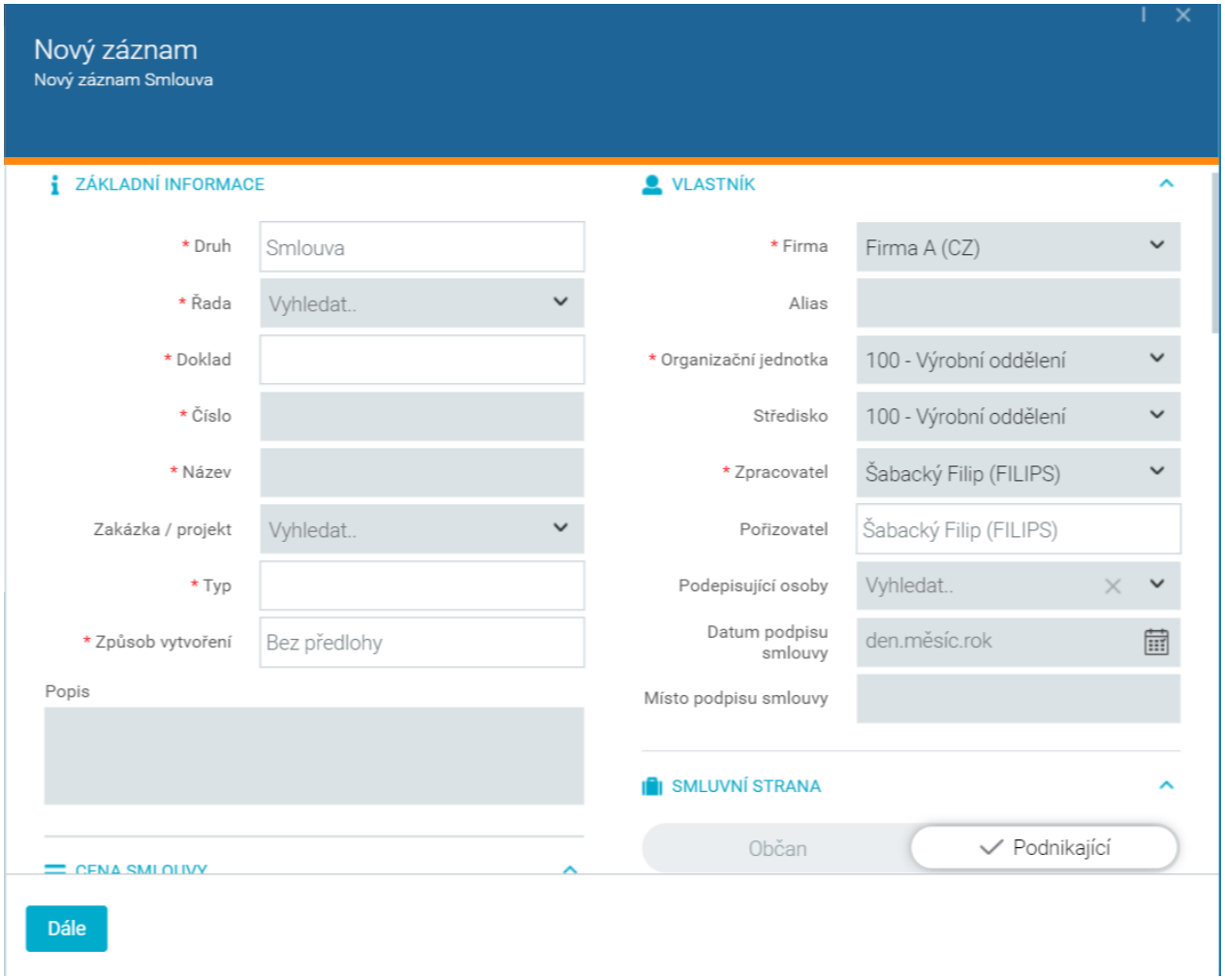
E - Rychlý přehled má uživateli poskytnout všechny základní údaje, které jsou v daném kontextu důležité. V mnoha případech je zde obsažen nějaký graf, který slouží k rychlejší orientaci a přehlednosti. Více informací je zobrazeno v panelu Detailu.

F - Aplikace myTEAM® poskytuje možnost vyhledávání nad všemi záznamy, které se v aplikaci nacházejí. Jelikož systém obsahuje nespočet velkých dat, je toto vyhledávání realizováno vyhledávačem Elasticsearch.

G - Pro mnoho akcí se v aplikaci využívají notifikace, které uživatele informují například o stavu nahrávaného dokumentu, odeslání zásilky nebo vygenerování různých dokumentů.

A.1 Ukázka formuláře aplikace myTEAM®

Velmi důležitou součástí aplikace jsou formuláře, které slouží pro založení a editaci záznamů. Na obrázku A.2 je formulář pro založení nové smlouvy. Obsahuje více různých komponent, které slouží například pro zadání čísla, krátkého a dlouhého textu, vysouvací nabídku, ve které lze zvolit více hodnot nebo přepínač mezi dvěma stavy.



Obrázek A.2: Ukázka formuláře smluv aplikace myTEAM®